jline 2.x in Action

2019年07月14日 12:49:00

我们都知道，软件的用户界面无非分为 GUI （图形用户界面）和 CLI （命令行用户界面）。对于我们经常使用 Linux 的人来说，命令行界面一定非常熟悉。无论是 Shell 里输入命令的界面，还是如 GDB 等软件的内部交互界面，都是命令行界面。而当我们开发自己的软件，要写认真写一个 CLI 的时候，却发现要手写做出一个好用的命令行界面其实非常困难。因为一个好的命令行界面，在输入/输出之外，还要支持一些常见的命令行功能。

对我而言，一个合格的命令行软件界面应该支持这三个功能：

* 自动补全：当按下 TAB 键时，在当前光标处进行内容补全。根据上下文信息，补全可能是对命令的补全，也可能是对文件路径的补全。
* 命令历史：当按上/下方向键时，可以显示上一条/下一条命令。
* 行编辑 (line editing)：可以使用 Emacs 快捷键进行行内的编辑功能，例如 Ctrl+A 移动光标至行首，Ctrl+E 移动光标至行尾。

熟悉 Linux 的人会发现，上面这三个功能都是 https://en.wikipedia.org/wiki/GNU\_Readline 的功能。我们不需要在软件中手写这几个功能，只要用这样一个库就可以了。实际上，GNU/Linux 中使用 GNU Readline 库的软件非常多，这使得 GNU Readline 同时也成为了一个事实上的命令行交互标准。GNU Readline 是 C 语言的库。我们用其他语言的时候，就要找对应功能的库（这往往是封装了底层的 GNU Readline 的库）。对 Java 语言来说， https://github.com/jline/jline2 就是这样一个帮助你搭建一个命令行交互界面的库。

本文是想通过一个例子介绍 Jline2 的基本用法。Jline2 并没有一个 “Hello, world!” 的例子，它的 https://github.com/jline/jline2/wiki 也写得非常简略。虽然有一个示例的程序  ，但这个示例比较复杂，难以理解。希望本文的内容能对你理解 Jline2 的用法有所帮助。

目录

1. jline 在Eclipse下运行的问题1

2. Jline 简单例子2

3. Jline细节讲解3

4. 简单补全与复合补全4

5. 命令历史5

6. 按键功能6

总结7

1. **jline 在Eclipse下运行的问题**

**问题描述**：在Eclipse下运行时，控制台无法接受输入，输入命令、回车，系统没反应。(先解决该问题后续代码才能跑得通)

**原因**：在通过JNI调用本地WindowsAPI时，本地的private static final native int ReadConsoleInputW();方法返回了非预期的值。导致控制台读取部分实际上处于死循环状态。

**解决方法**：设置系统属性jline.WindowsTerminal.directConsole为false。（绕过本地库Windows API调用，直接使用Java IO流读取输入）System.setProperty("jline.WindowsTerminal.directConsole", "false");

1. **Jline 简单例子**

**package** com.citicbank.bdp.jline\_example;

**import** java.io.File;

**import** java.util.ArrayList;

**import** java.util.List;

**import** jline.console.ConsoleReader;

**import** jline.console.completer.AggregateCompleter;

**import** jline.console.completer.ArgumentCompleter;

**import** jline.console.completer.Completer;

**import** jline.console.completer.FileNameCompleter;

**import** jline.console.completer.NullCompleter;

**import** jline.console.completer.StringsCompleter;

**import** jline.console.history.FileHistory;

**public** **class** App

{

**public** **static** **void** main( String[] args )

{

// 解决在Eclipse下运行时，控制台无法接受输入，输入命令、回车，系统没反应

System.*setProperty*("jline.WindowsTerminal.directConsole", "false");

// 是否打印调试信息

System.*setProperty*("jline.internal.Log.debug", "false");

// 按下无效按键时不响铃

System.*setProperty*("jline.nobell", "false");

String line = "";

**try** {

ConsoleReader reader = **new** ConsoleReader();

/\*\*

\* MemoryHistory: 命令历史保存在内存中;

\* FileHistory : 命令历史保存在文件中。

\* 执行完命令必须执行history.flush();才能保存到文件中，

\* reader.setHistory(history);只是从文件中加载历史.

\*/

// History history = new MemoryHistory();

FileHistory history = **new** FileHistory(**new** File("/opt/source/jline-example/history/history.txt"));

reader.setHistory(history);

// 开启该参数，历史命令才能生效

reader.setHistoryEnabled(**true**);

// 设置提示字符，也可以在readLine中指定，如: line = reader.readLine(" hive > ");

reader.setPrompt(" hive > ");

//console.setBellEnabled(true);

//console.setExpandEvents(false);

/\*\*

\* 自动补全

\* 1. StringsCompleter

\* 2. FileNameCompleter

\* 3. NullCompleter

\* 4. 自定义

\* 5. ArgumentCompleter 集合类

\* 必须通过该类，不能直接reader.addCompleter以上类，否则只会实现其中之一

\* 多个ArgumentCompleter可以用AggregateCompleter聚合

\*/

Completer strCompleter = **new** StringsCompleter(**new** String[]{"create", "drop", "insert"});

Completer strCompleter2 = **new** StringsCompleter(**new** String[]{"into", "table", "template"});

Completer myCompleter = **new** MyCompleter();

Completer fileCompleter = **new** FileNameCompleter();

Completer nullCompleter = NullCompleter.***INSTANCE***;

List<Completer> completerList = **new** ArrayList<Completer>();

completerList.add(strCompleter);

completerList.add(strCompleter2);

completerList.add(myCompleter);

completerList.add(fileCompleter);

completerList.add(nullCompleter);

reader.addCompleter(**new** ArgumentCompleter(completerList));

// reader.addCompleter(new AggregateCompleter(arg1, arg2));

**while**(**true**){

// line = reader.readLine("hive > ");

// line = reader.readLine('\*');

line = reader.readLine();

//System.out.println("jline output > " + line);

**if** (!"".equals(line.trim())){

System.***out***.println("INFO : you are exec " + line + "!");

}

history.flush();

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

1. **Jline细节讲解**

java的控制台输入是很局限的，我们仅仅可得到它的输入，对于输入的过程是很难操控的，所以当我们想写一个人性化的输入体验的时候，是比较难实现的。还好java中有种jni的技术，它允许Java代码和其他语言写的代码进行交互。

Jline就是一个使用了C/C++实现的java类库，它可以让你更方便的处理控制台输入。  
引入Jline，使用下面的maven配置

|  |  |
| --- | --- |
|  | **<dependency>**  **<groupId>**jline**</groupId>**  **<artifactId>**jline**</artifactId>**  **<version>**2.9**</version>**  **</dependency>** |

我们平常处理输入一般是这样的：

|  |  |
| --- | --- |
|  | BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.in));  String line = **null**;  **do**  {  line = br.readLine();  **if**(line != **null**) {  *//todo*  }  }**while**(line!=**null** && !line.equals("exist")) |

在使用了Jline之后，一样的书写形式：

|  |  |
| --- | --- |
|  | ConsoleReader reader = **new** ConsoleReader();  String line = **null**;  **do** {  line = reader.readLine("input>");  **if**(line != **null**){  *//TODO*  }  }  **while**(line!=**null** && !line.equals("exist")) |

你便拥有了一个比较友好的输入体验了。它可以实现光标的移动，通过上下方向键切换历史命令等操作。  
如果你希望有更强大的自动补全功能。你可以给ConsoleReader 设一个jline.console.completer.Completer接口实现类(下一节详细讲解命令补全)

|  |  |
| --- | --- |
|  | **public** **interface** Completer  **int** complete(String buffer, **int** cursor, List<CharSequence> candidates);  } |

其中buffer是当前用户输入的内容，cursor表示光标的位置，candidates表示你想补全的候选项。返回值很重要，表示你要在哪个位置补全你的内容，假设用户的输入是ls my xxxxxx  
目前用户的光标在my后面，你想帮用户补全为myfoler或者myfile。你要这么写：

|  |  |
| --- | --- |
|  | **public** **class** MyCompleter **implements** Completer {  @Override  **public** **int** complete(String buffer, **int** cursor, List candidates)  {  **if**(buffer.length() > 0 && cursor >0)  {  String substring = buffer.substring(0,cursor);  **if**(substring.endsWith("my"))  {  candidates.add("myfolder");  candidates.add("myfile");  **return** cursor-2;  }  }  **return** cursor;  }  } |

然后指定给comsoleReader对象即可：

|  |  |
| --- | --- |
|  | MyCompleter myCompleter = **new** MyCompleter();  reader.addCompleter(myCompleter); |

如果你的candidates有多个候选项，它会帮你列举出来，如果只有一个，它会帮你自动补全，当你的候选项都是xxx开头的，它会帮你补全一部分。  
功能和vi，eclipse等ide里的自动补全功能一样强大。

Jline还有好多功能，例如获取历史，快捷键，移动光标，清屏等功能，它很强大，你只要知道有这么个东西，再去看看代码，看看它的成员方法，什么都是可以解决的。

// 设置提示字符，也可以在readLine中指定，如: line = reader.readLine(" hive > ");

reader.setPrompt(" hive > ");

这里设置命令提示符 (prompt)，没有进行任何特殊的设置。命令行会将用户输入的一行原样打印出来。当用户输入 Ctrl+D (End of line) 时，程序会退出。

1. **简单补全与复合补全**

由于命令补全和程序的命令格式密切相关，所以我们必须自己定义补全的方式。Jline2 中定义命令补全的方式是：创建一个 Completer 类的实例，将其传入 LineReader 。Jline2 内置了多个 completer，其中最常见的是 FileNameCompleter （补全文件名）和 StringsCompleter （根据预定义的几个字符串进行补全，用于命令名或参数名）。例如，程序的四个命令分别以 CREATE, OPEN, WRITE, CLOSE 开头，那么我们可以使用一个 StringsCompleter来对命令的第一个单词进行补全：

**Completer** commandCompleter = new StringsCompleter("CREATE", "OPEN", "WRITE", "CLOSE");

**ConsoleReader** reader **=** new **ConsoleReader();**

reader.addCompleter(commandCompleter);

然而，这种补全方式只能支持每个命令的第一个单词，我们想要在命令的各种可能的地方都进行补全该怎么办呢？这时候就需要将 completer 进行组合，形成 **复合 completer** 。一般情况下，StringsCompleter 这样的 **简单 completer** 只能负责一个单词的补全，而要想实现整条命令的补全，就需要将几个不同的 completer 组合起来使用。 ArgumentCompleter 就是用来补全整条命令的复合 completer。它可以将若干个 completer 组合在一起，每个 completer 负责补全命令中的第 i 个单词。以 CREATE 命令为例，这条命令共有两个单词，第一个单词需要字符串补全，第二个单词需要文件名补全。于是我们使用 ArgumentCompleter 将 StringsCompleter 和 FileNameCompleter 组合起来：

Completer createCompleter = **new** ArgumentCompleter( **new** StringsCompleter("CREATE"), **new** FileNameCompleter());

**ConsoleReader** reader **=** new **ConsoleReader();**

reader.addCompleter(createCompleter);

根据 ArgumentCompleter 的两个参数，在输入第一个单词的时候会补全 CREATE，输入第二个单词的时候会补全文件名。但实测时会发现一个问题：当你已经输入了 CREATE 和文件名后，再试图进行补全，在第三个单词处试图补全，还是会出现文件名的补全。这是因为， ArgumentCompleter 在你已经“用完了”所有的 completers 之后（即第三个单词开始），会默认使用最后一个 completer。这并不是我们想要的效果。为了解决这个问题，我们可以在最后添加一个 NullCompleter：

Completer createCompleter = **new** ArgumentCompleter( **new** StringsCompleter("CREATE"), **new** FileNameCompleter(), NullCompleter.INSTANCE);

**ConsoleReader** reader **=** new **ConsoleReader();**

reader.addCompleter(createCompleter);

NullCompleter 即不进行任何补全。这样，从第三个单词开始，都不会进行任何多余的补全。

类似地，我们再加入 OPEN 命令补全的定义：

Completer createCompleter = new ArgumentCompleter( new StringsCompleter("CREATE"), new FileNameCompleter(), NullCompleter.INSTANCE);

Completer openCompleter = new ArgumentCompleter( new StringsCompleter("OPEN"), new FileNameCompleter(), new StringsCompleter("AS"), NullCompleter.INSTANCE);

Completer aggCompleter = new AggregateCompleter(createCompleter, openCompleter);

**ConsoleReader** reader **=** new **ConsoleReader();**

reader.addCompleter(aggCompleter);

这里有两点需要注意的地方：

CREATE 命令和 OPEN 命令分别定义了 completer，再用 AggregateCompleter 组合起来。 AggregateCompleter 是另一种复合 completer，将多种可能的补全方式组合到了一起。打比方来说， ArgumentCompleter 相当于串联电路， AggregateCompleter 相当于并联电路。OPEN 命令的 ArgumentCompleter 中只定义了前三个单词的补全方式。这是因为第四个单词是用户定义了文件变量，用户可能输入任何的名字，因此无法进行补全。

1. **命令历史**

前面已经过说，在默认情况下，Jline2 已经支持命令历史查找。不过我们想加上一个特殊的功能：用户输入的注释（以 # 开头）不会进入命令历史，从而在命令历史查找时不受注释内容的干扰。

JLine3 中， History 负责控制历史记录的行为 。查看源代码，我们发现 add() 方法是其核心行为。用户输入的一行命令，会通过 add() 方法加入命令历史中。

@Override

**public** **void** **add**(Instant time, String line) {

Objects.requireNonNull(time);

Objects.requireNonNull(line);

**if** (getBoolean(reader, LineReader.DISABLE\_HISTORY, **false**)) {

**return**;

}

// ... internalAdd(time, line);

// ...

}

同样地，我们可以通过继承并重写 add() 方法，将注释内容过滤掉，不加入命令历史：

**public** **final** **class** MyHistory extends History {

**private static boolean** isComment(**String** line) {

return line.startsWith("#");

}

@Override

**public void** add(**Instant** time, **String** line) {

if (isComment(line)) {

return;

}

super.add(time, line);

}

}

然后我们这样设置 LineReader ：

ConsoleReader reader = new ConsoleReader();

reader.setHistory(history);

1. **按键功能**

// Control keys.

Operation.SET\_MARK, /\* Control-@ \*/

Operation.BEGINNING\_OF\_LINE, /\* Control-A \*/

Operation.BACKWARD\_CHAR, /\* Control-B \*/

Operation.INTERRUPT, /\* Control-C \*/

Operation.EXIT\_OR\_DELETE\_CHAR, /\* Control-D \*/

Operation.END\_OF\_LINE, /\* Control-E \*/

Operation.FORWARD\_CHAR, /\* Control-F \*/

Operation.ABORT, /\* Control-G \*/

Operation.BACKWARD\_DELETE\_CHAR, /\* Control-H \*/

Operation.COMPLETE, /\* Control-I \*/

Operation.ACCEPT\_LINE, /\* Control-J \*/

Operation.KILL\_LINE, /\* Control-K \*/

Operation.CLEAR\_SCREEN, /\* Control-L \*/

Operation.ACCEPT\_LINE, /\* Control-M \*/

Operation.NEXT\_HISTORY, /\* Control-N \*/

null, /\* Control-O \*/

Operation.PREVIOUS\_HISTORY, /\* Control-P \*/

Operation.QUOTED\_INSERT, /\* Control-Q \*/

Operation.REVERSE\_SEARCH\_HISTORY, /\* Control-R \*/

Operation.FORWARD\_SEARCH\_HISTORY, /\* Control-S \*/

Operation.TRANSPOSE\_CHARS, /\* Control-T \*/

Operation.UNIX\_LINE\_DISCARD, /\* Control-U \*/

Operation.QUOTED\_INSERT, /\* Control-V \*/

Operation.UNIX\_WORD\_RUBOUT, /\* Control-W \*/

emacsCtrlX(), /\* Control-X \*/

Operation.YANK, /\* Control-Y \*/

null, /\* Control-Z \*/

emacsMeta(), /\* Control-[ \*/

null, /\* Control-\ \*/

Operation.CHARACTER\_SEARCH, /\* Control-] \*/

null, /\* Control-^ \*/

Operation.UNDO, /\* Control-\_ \*/

## 总结

我们发现，Jline2.x 的各个功能设计得比较清晰，有其对应的接口和默认实现。如果我们想自定义一些特性，一般通过继承并重写的方式可以做到。Jline2.x 的源代码也比较容易理解，遇到困难时，可以自己阅读源代码来寻找线索。